105 透彻研究通过explain命令得到的SQL执行计划（6）

今天我们继续来讲解执行计划的一些细节，之前已经详细讲过了select_type和type，今天来先讲一下possible_keys

这个possible_keys，顾名思义，其实就是在针对一个表进行查询的时候有哪些潜在可以使用的索引。

比如你有两个索引，一个是KEY(x1, x2, x3)，一个是KEY(x1, x2, x4)，此时要是在where条件里要根据x1和x2两个字段进行查询，那么此时明显是上述两个索引都可以使用的，那么到底要使用哪个呢？

此时就需要通过我们之前讲解的成本优化方法，去估算使用两个索引进行查询的成本，看使用哪个索引的成本更低，那么就选择用那个索引，最终选择的索引，就是执行计划里的key这个字段的值了。

而key_len，其实就是当你在key里选择使用某个索引之后，那个索引里的最大值的长度是多少，这个就是给你一个参考，大概知道那个索引里的值最大能有多长，就这么个意思。

而执行计划里的 ref 也相对会关键一些，当你的查询方式是索引等值匹配的时候，比如const、ref、eq_ref、ref_or_null这些方式的时候，此时执行计划的ref字段告诉你的就是：你跟索引列等值匹配的是什么？是等值匹配一个常量值？还是等值匹配另外一个字段的值？

比如SQL语句：

EXPLAIN SELECT * FROM t1 WHERE x1 = 'xxx'

此时如果你看他的执行计划是下面这样的

```
+----+-------------+-------+------------+------+---------------+----------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key      | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+----------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | t1    | NULL       | ref  | index_x1      | index_x1 | 589     | const |  468 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+----------+---------+-------+------+----------+-------+
```

大家在上面的查询计划里可以看到，针对t1表的查询，type是ref方式的，也就是说基于普通的二级索引进行等值匹配，然后possible_keys只有一个，就是index_x1，针对x1字段建立的一个索引，而实际使用的索引也是index_x1，毕竟就他一个是可以用的。

然后key_len是589，意思就是说index_x1这个索引里的x1字段最大值的长度也就是589个字节，其实这个不算是太大，不过基本可以肯定这个x1字段是存储字符串的，因为是一个不规律的长度。

比较关键的是ref字段，它的意思是说，既然你是针对某个二级索引进行等值匹配的，那么跟index_x1索引进行等值匹配的是什么？是一个常量或者是别的字段？这里的ref的值是const，意思就是说，是使用一个常量值跟index_x1索引里的值进行等值匹配的。

假设你要是用了类似如下的语句：

EXPLAIN SELECT * FROM t1 INNER JOIN t2 ON t1.id = t2.id;

此时执行计划里的ref肯定不是const，因为你跟t1表的id字段等值匹配的是另外一个表的id字段，此时ref的值就是那个字段的名称了，执行计划如下：

```
+----+-------------+-------+------------+--------+---------------+---------+---------+-----------------+------+----------+-------+
| id | select_type | table | partitions | type   | possible_keys | key     | key_len | ref             | rows | filtered | Extra |
+----+-------------+-------+------------+--------+---------------+---------+---------+-----------------+------+----------+-------+
|  1 | SIMPLE      | t1    | NULL       | ALL    | PRIMARY       | NULL    | NULL    | NULL            | 3457 |   100.00 | NULL  |
|  1 | SIMPLE      | t2    | NULL       | eq_ref | PRIMARY       | PRIMARY | 10      | test_db.t1.id |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+--------+---------------+---------+---------+-----------------+------+----------+-------+
```

大家看执行计划，针对t1表作为驱动表执行一个全表扫描，接着针对t1表里每条数据都会去t2表根据t2表的主键执行等值匹配，所以第二个执行计划的type是eq_ref，意思就是被驱动表基于主键进行等值匹配，而且使用的索引是PRIMARY就是使用了t2表的主键。

至于ref，意思就是说，到底是谁跟t2表的聚簇索引里的主键值进行等值匹配呢？是常量值吗？

不是，是test_db这个库下的t1表的id字段，这里跟t2表的主键进行 等值匹配的是t1表的主键id字段，所以ref这里显示的清清楚楚的。

最后简单说一下rows和filtered，这个rows顾名思义，就是说你使用指定的查询方式，会查出来多少条数据，而filtered意思就是说，在查询方式查出来的这波数据里再用上其他的不在索引范围里的查询条件，又会过滤出来百分之几的数据。

比如SQL语句：

EXPLAIN SELECT * FROM t1 WHERE x1 > 'xxx' AND x2 = 'xxx'

他只有一个x1字段建了索引，x2字段是没有索引的，此时执行计划如下：

```
+----+-------------+-------+------------+-------+---------------+----------+---------+------+------+----------+-------------------------
----------------+
```

| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |

+----+-------------+-------+------------+-------+---------------+----------+---------+------+------+----------+-----------------------+

| 1 | SIMPLE | t1 | NULL | range | index_x1 | index_x1 | 458 | NULL | 1987 | 13.00 | Using index condition; Using where |

+----+-------------+-------+------------+-------+---------------+----------+---------+------+------+----------+-----------------------+

上面的执行计划清晰的表明了，针对t1表的查询方式是range，也就是基于索引进行范围查询，用的索引是index_x1，也就是x1字段的索引，然后基于x1>'xxx'这个条件通过index_x1索引查询出来的数据大概是1987条，接着会针对这1987条数据再基于where条件里的其他条件，也就是x2='xxx'进行过滤。

这个filtered是13.00，意思是估算基于x2='xxx'条件过滤后的数据大概是13%，也就是说最终查出来的数据大概是1987 * 13% = 258条左右。

好，今天执行计划分析就到这里，其实大家看到这里为止，基本上对于执行计划已经了解的很清楚了，接下来下次就是执行计划分析的最后一讲，也就是分析extra这个字段，这里会包含了各种查询的附加条件，也是非常重要的

看懂了extra之后，我们以后对任何SQL语句的执行计划都能逐步分析得到结论，知道这个SQL语句是如何一步一步执行的了，过程中每个步骤查询出来多少条数据。

**End**